Parma University, Math. Department

Algorithms and Data Structures
Open Hashing

Marco Pellegrini

Summary of lecture 11

- (CLR 12.4)
- hashing with open addressing,
- insertion/deletion
- linear and quadratic probing
- re-hashing

Open addressing.

- The idea of open addressing is to store the lists of elements with the same hash value inside the hash table T[1..m] instead of outside as linked lists.
- We saw in lecture 2 that we can simulate a linked list inside an array, but we needed to store pairs (next, data).
- Here we want to compute the addresses instead of storing (and following) pointers.
- Naturally if we hash to a used slot we have to try an other slot until we find an empty one. Each try is called a "probe".

Probes.

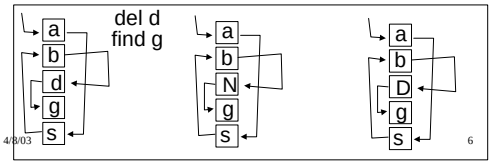
- We have the universe of keys U,
  the possible probes P = {0,...,m-1}
  the addresses in the table A={0,..m-1}.
- The hashing function in open addressing is
  h: U x P ---> A
  That is it takes as input the key k, the probe number p, and returns an address h(k,p).
- We will require that probe sequence visits all the addresses, for any key k,
  {h(k,0),...,h(k,m-1)} = {0,...m-1}.

Insertion

- The array T[0,..,m-1] holds a key or NIL if the location is unused.
- ```
  Hash-insert(T,k)
   i := 0
   REPEAT j := h(k,i)
     IF T[j] = NIL
        THEN T[j] := k, RETURN j
     END IF
   i := i +1
  UNTIL i =m
  error "table overflow"
  ```
- Exercise: write a code for searching k in T assuming that we do not delete from the table.

How to handle deletions.

- We can use the code for searching an element to find it and then delete it. But we cannot simply place NIL in its place since this will break a chain of probes.
- We use a special marker D=DELETED instead. During insertion we write onto a DELETED position. During searching we skip over a DELETED position.

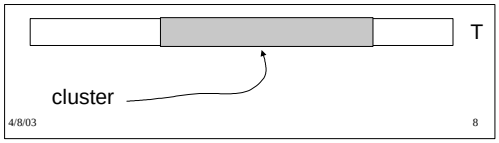

  ```

Observations

- If in a table we have elements, Nil=N, and DELETED=D then the performance of the operations does not depend only on the load factor (number of elements)/m. This makes analysis more complicated.
- The ideal situation is when every probe sequence is as likely as another. This is called Uniform hashing hypothesis.
- Formally, if k is taken randomly and uniformly from U then the sequence [h(k,0),...h(k,m-1)] has probability 1/m! of being any of the permutations of [0,..,m-1].
- True uniform hashing is hard to get, so heuristic methods are used.

Linear probing

- Linear probing is the idea to use normal hashing initially and then start looking ahead one position at a time.
- Formally: h: U $\times$[0,m-1]$\rightarrow$[0,m-1] and h':U $\rightarrow$[0,m-1] .
- h(k,i) = (h'(k) + i) mod m.
- Linear probing suffers the problem of primary clusters, that is, if there is a long cluster it tends to grow even longer.



Quadratic probing

- $h(k,i) = (h'(k) + c_1 i + c_2 i^2) \bmod m$
- Quadratic probing avoids primary clustering because at every probe the step becomes larger and larger.
- It still has a secondary clustering, that is if $h(k_1,0) = h(k_2, 0)$ then the probe sequence is the same.
- To reach all of T, the values c_1, c_2 and m must be chosen carefully.

Double hashing

- This is the method that gets closer to the uniform hashing behavior.
- we need two normal hashing functions h_1 and h_2
- $h'(k,i) = (h_1(k) + i h_2(k)) \bmod m$
- Now both the initial probe and the probe step depend on both k.
- In order for the probe sequence to visit the whole table $h_2(k)$ and m must be relative primes.
- We can either (1) choose m prime;
- (2) choose m a power of 2 and h_2 to return only
- odd numbers.

Exercise

- CLR 12.4-1. Insert the keys 10, 22, 31, 4, 15, 28 into an hash table with m= 11, using open addressing. The primary hash function is h'(k) = k mod m. Use linear probing.
  use quadratic probing with c_1= 1 and c_2=3
  use double hashing with $h_2(k) = 1 + (k \bmod (m-1))$.

Analysis of open-address hashing

- We have a table of m slots holding n keys (without slots marked D).
- We want to bound the search time by a function of the load factor n/m = a < 1
- We assume uniform hashing.
- Theorem 1 (CLR 12.5)
  The expected number of probes searching for an element *not in* the table is at most 1/(1-a).
- Theorem 2 (CLR 12.7)
  The expected number of probes searching for an element *in* the table is at most (1/a)[1+ln(1/(1-a))].

Exercise (CLR 12.4-4)

- Consider an open-address table hash table with uniform hashing and load factor 1/2.
  (1) what is the expected time to search for an element present in the table?
  (2) what is the expected time to search for an element not in the table?
  (3) Repeat the calculations for load factors of 3/4 and 7/8.

Proof of theorem 1.

- Let X be a variable that can assume values 0,1,..
- X will be the number of times we have to repeat probing.
- The expected value of X is by definition: $E[X] = \sum_i i \, Prob\{X=i\}$
- But $Prob\{X=i\} = (Prob\{X \geq i\} - Prob\{X \geq i+1\})$
- Substituting $E[X] = \sum_i i(Prob\{X \geq i\} - Prob\{X \geq i+1\})$.
- In the summation $Prob\{i+1\}$ appears twice, once multiplied by -i, once by i+1, so we get
- $E[X] = \sum_{i>0} Prob\{X \geq i\}$

- $Prob\{X \geq 1\}$ is the probability that we have to repeat probing more then once, that is the probability that the first location we probe is used = n/m=a.
- $Prob\{X \geq 2\}$ is the probability that we have to repeat probing at least twice = probability that the first two location we probe are occupied.
- This is $(n/m)((n-1)/(m-1)) \leq a^2$.
- In general $Prob\{X \geq i\}$ is the probability that we find the first i slots used when we probe, that is $(n/m)((n-1)/(m-1))....((n-i)/(m-i)) \leq a^i$.
- So the expected number of times we have to hash is: $1 + E[X] \leq 1 + a + a^2 + a^3 + ... + a^i \leq 1/(1-a)$ when $a < 1$.

Proof of Theorem 2.

- When we search for a key k that is in T and was inserted when in the table T there were i elements, than we probe the same elements that were probed when k was inserted.
- But before inserting k, k was not there so the number of elements accessed was at most $1/(1 -(i/m))= m/(m-i)$ by Theorem 1.
- Now we take the average on all the n keys in T. That is

$$\frac{1}{n}\sum_{i=0}^{n-1}\frac{m}{m-i} = \frac{m}{n}\sum_{i=0}^{n-1}\frac{1}{m-i} = \frac{1}{a}(H_m - H_{m-n})$$

where H_j is the j-th harmonic number

Harmonic numbers follow closely the natural logarithm (base e= 2.7..). In particular: $\ln i \leq H_i \leq \ln i + 1$

So substituting in the previous expression we get that it is $\leq (1/a)(1 + \ln m - \ln (m-n))$.

The difference of two logarithms is the logarithm of the ratio of the arguments so we get:

$$(1/a)(1 + \ln (m/(m-n)) = (1/a)(1 + \ln (1/(1-a))).$$

Homework

CLR 12.4-2. Write pseudocode for the operations Delete(k), Insert(k) and Search(k) for open addressing taking into account that a slot can hold an element, NIL or the DELETED flag.

Conclusion

- We have seen a method to store all keys inside an hash table (open-addressing).
- The method is based on repeated probing to find an element or a free slot. Probing can be done by linear probing, quadratic probing or double hashing.
- We have derived bounds on the time in terms of the load factor (without deletions).